# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **02/1/2025** | **John Zephyr** |  |

## Client



## “Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.”

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also include images or support materials. If you include them, make certain to insert them in the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.”

## Developer

John Zephyr

1. Interpreting Client Needs

Determine your client’s needs and potential threats and attacks associated with the company’s application and software security requirements. Consider the following questions regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

Artemis Financial specializes in developing comprehensive financial plans for individuals, addressing areas such as savings, retirement, investments, and insurance. Secure communication is critical, as the company manages sensitive client information, including Social Security Numbers and tax data. A key regulatory consideration for secure communications is ensuring that trade secrets remain protected and undisclosed. Given the pressing need to safeguard all types of information, the main external threat involves potential breaches targeting client data. Consequently, robust encryption of any data accessed by unauthorized individuals is imperative. Furthermore, keeping security measures up to date with regular bug fixes and addressing vulnerabilities must be a top priority for Artemis Financial.

**“2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.”

* Input Validation: To validate the account user’s login information. This would provide protection for users.
* Code Quality: It is essential to manage access methods based on user roles within code quality. Each user should only have access to their information, thereby preventing unauthorized access to other user's data and the server.
* APIs: The development of an API is crucial for both internal and external operations. This will establish guidelines for acceptable data access and interactions.
* Code Errors: Implementing effective error handling is important, as it enables the identification of areas within the API that require attention and resolution. Consequently, Artemis Financial can ensure that user information remains secure and is protected from potential exposure or unauthorized access.
* Cryptography: The incorporation of cryptography at Artemis Financial is vital for safeguarding user information. Given the involvement of various currencies and global transactions, employing cryptographic techniques will enhance the security of user data against potential breaches.

**“3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.”

They did not avoid methods that expose sensitive information, like account numbers, directly. It would be better to use more secure methods to both handle and display customer data. The class was declared public, which means it is accessible from any other package. It would have been more conventional to use PascalCase (e.g., Customer) for the class name, which would also have followed Java's naming conventions. The account\_number field was correctly marked as private, which ensures proper encapsulation. It might be beneficial, however, to add a getter and setter for this field. Doing so would allow you to control the access and validate the account number.

The account balance field was declared as public. However, this exposes it directly and could lead to unintended modifications. It would be better to mark it as private and add appropriate getter and setter methods. The showInfo method returns the account number, which is not the best practice in a real-world application, given privacy concerns. Need to return more comprehensive customer information or use other methods to manage sensitive data securely.

**“4. Static Testing**

**Run a dependency check on Artemis Financial’s software application to identify all** **security vulnerabilities in the code. Record the output from the dependency-check** **report. Include the following** items:

* **The names or vulnerability codes of the known vulnerabilities**
* **A brief description and recommended solutions provided by the dependency-check** report.
* Any **attribution that documents how this vulnerability has been identified or** **documented previously**.”

CVE-2016-1000352

In the Bouncy Castle JCE Provider version 1.55 and earlier the ECIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider.

V3.0: 7.4 HIGH

V2.0: 5.8 MEDIUM

CVE-2016-1000346

In the Bouncy Castle JCE Provider version 1.55 and earlier the other party DH public key is not fully validated. This can cause issues as invalid keys can be used to reveal details about the other party's private key where static Diffie-Hellman is in use. As of release 1.56 the key parameters are checked on agreement calculation.

V3.0: 3.7 LOW

V2.0: 4.3 MEDIUM

CVE-2016-1000345

In the Bouncy Castle JCE Provider version 1.55 and earlier the DHIES/ECIES CBC mode vulnerable to padding oracle attack. For BC 1.55 and older, in an environment where timings can be easily observed, it is possible with enough observations to identify when the decryption is failing due to padding.

V3.0: 5.9 MEDIUM

V2.0: 4.3 MEDIUM

CVE-2016-1000344

In the Bouncy Castle JCE Provider version 1.55 and earlier the DHIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider.

V3.0: 7.4 HIGH

V2.0: 5.8 MEDIUM

CVE-2016-1000343

In the Bouncy Castle JCE Provider version 1.55 and earlier the DSA key pair generator generates a weak private key if used with default values. If the JCA key pair generator is not explicitly initialized with DSA parameters, 1.55 and earlier generates a private value assuming a 1024-bit key size. In earlier releases this can be dealt with by explicitly passing parameters to the key pair generator.

V2.0: 5.0 MEDIUM

CVE-2016-1000342

In the Bouncy Castle JCE Provider version 1.55 and earlier ECDSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure.

V3.0: 7.5 HIGH

V2.0: 5.0 MEDIUM

CVE-2016-1000341

In the Bouncy Castle JCE Provider version 1.55 and earlier DSA signature generation is vulnerable to timing attack. Where timings can be closely observed for the generation of signatures, the lack of blinding in 1.55, or earlier, may allow an attacker to gain information about the signature's k value and the private value as well.

V3.0: 5.9 MEDIUM

V2.0: 4.3 MEDIUM

CVE-2016-1000339

In the Bouncy Castle JCE Provider version 1.55 and earlier the primary engine class used for AES was AESFastEngine. Due to the highly table-driven approach used in the algorithm it turns out that if the data channel on the CPU can be monitored the lookup table accesses are sufficient to leak information on the AES key being used. There was also a leak in AESEngine although it was less. AESEngine has been modified to remove any signs of leakage (testing conducted on Intel X86-64) and is now the primary AES class for the BC JCE provider from 1.56. Use of AESFastEngine is now only recommended where otherwise deemed appropriate.

V3.0: 5.3 MEDIUM

V2.0: 5.0 MEDIUM

CVE-2016-1000338

In Bouncy Castle JCE Provider version 1.55 and earlier the DSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validated, which in some cases may allow the introduction of 'invisible' data into a signed structure.

CVE-2023-20883

In Spring Boot versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14 and older unsupported versions, there is potential for a denial-of-service (DoS) attack if Spring MVC is used together with a reverse proxy cache.

V3.1: 7.5 HIGH

V2.0:(not available)

CVE-2023-20873

In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+.

V3.1: 9.8 CRITICAL

CVE-2022-27772

Spring-boot versions prior to version v2.2.11.RELEASE was vulnerable to temporary directory hijacking. This vulnerability impacted the org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir method. NOTE: This vulnerability only affects products and/or versions that are no longer supported by the maintainer.

**5. Mitigation Plan**

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financials’ software application.

Interpreting the results:

* CVE-2016-1000352 & CVE-2016-1000344: Both vulnerabilities relate to the use of ECB mode in ECIES implementation within Bouncy Castle JCE Provider versions 1.55 and earlier. ECB mode is considered unsafe due to its lack of diffusion, making it susceptible to various attacks.
* CVE-2016-1000346: This vulnerability involves insufficient validation of the other party's DH public key in Bouncy Castle JCE Provider versions 1.55 and earlier, which could lead to leakage of private key details under certain conditions.
* CVE-2016-1000345 & CVE-2016-1000339: These vulnerabilities concern weaknesses in AES and DSA implementations that can be exploited through padding oracle attacks or side-channel analysis.
* CVE-2016-1000343 & CVE-2016-1000341: Both vulnerabilities relate to timing attacks on DSA signature generation, where attackers could potentially gain information about private key values based on observed signature generation times.
* CVE-2016-1000342 & CVE-2016-1000338: These issues pertain to an inadequate validation of ASN.1 encoding in ECDSA and DSA signatures, allowing the injection of extra elements into signatures.
* CVE-2023-20883 & CVE-2023-20873: Both vulnerabilities are related to Spring Boot versions 3.0.0 - 3.0.6 and 2.7.0 - 2.7.11, involving potential denial-of-service (DoS) attacks through reverse proxy caching and security bypasses in Cloud Foundry deployments.
* CVE-2022-27772: This vulnerability affects Spring Boot versions prior to v2.2.11.RELEASE concerning temporary directory hijacking.

Mitigation Steps:

1. Switch to HTTPS protocol for all communications:
   * Ensure that all data transmission between clients and servers is encrypted using the HTTPS protocol.
   * Update server configurations to enforce HTTPS connections.
   * Implement SSL/TLS certificates and ensure they are up to date.
2. Move request parameters to headers or body rather than URI:
   * Avoid exposing sensitive information in URIs by moving request parameters to HTTP headers or request bodies.
   * Refactor API endpoints to use POST requests with JSON payloads for sensitive data.
   * Ensure that headers are also protected, especially when carrying authentication tokens.
3. Remove hard-coded database connection credentials:
   * Store database connection credentials securely and avoid hard-coding them in the application.
   * Use environment variables or configuration files to manage credentials.
   * Implement a secrets management solution if necessary.
4. Implement a secure authentication scheme:
   * Adopt a robust authentication mechanism such as OAuth 2.0, JWT (JSON Web Tokens), or another industry-standard method.
   * Ensure that tokens are securely generated and validated.
   * Use HTTPS to protect the transmission of authentication data.